UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Desarrollo de plataforma para modificación de modelos digitales tridimensionales de corazones.

Trabajo de graduación presentado por Christopher Valentin Ajú Tzay
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2019

Desarrollo de plataforma para modificación de modelos digitales tridimensionales de corazones.

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Desarrollo de plataforma para modificación de modelos digitales tridimensionales de corazones.

Trabajo de graduación presentado por Christopher Valentin Ajú Tzay
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2019

Vo.Bo.:

(f) _____
Ing. Dennis Aldana Moscoso

Tribunal Examinador:

(f) _____
Ing. Dennis Aldana Moscoso

(f) _____
Ing. Pablo Mazariegos de la Cerda

(f) _____
Ing. Kurt Kellner Juarez

Fecha de aprobación: Guatemala, 4 de diciembre de 2019.

Prefacio

Este trabajo se realizó con la idea de aplicar los conocimientos obtenidos durante la carrera para apoyar un proceso donde el tiempo puede representar la vida o la muerte de un ser humano. En el camino se encontraron dificultades que fueron superadas con el apoyo de personas que brindaron ideas o apoyo moral cuando más fue necesario.

Agradezco a mis padres, Valentin Ajú y Silvia Tzay por el apoyo durante estos años y las enseñanzas de vida.

Agradezco al Ingeniero Dennis Moritz Aldana, asesor en este proceso por la ayuda brindada, la cual hizo posible la realización de este proyecto.

Índice

Lista de figuras

Resumen

Este trabajo presenta la etapa de modificación de un modelo tridimensional digital. Esta etapa permite a los médicos observar y modificar el modelo digital 3D que se generó utilizando un examen médico, en la primera etapa, previo a realizar la impresión 3D. De esta manera se puede asegurar que el modelo tiene los detalles del órgano, necesarios para realizar la planificación de una operación quirúrgica. Este es uno de tres procesos necesarios para llevar una imagen médica del corazón a un modelo físico, los cuales son: la transformación del examen médico a un modelo tridimensional, la modificación y la impresión tridimensional del modelo.

Para poder realizar la modificación y la generación del archivo que contiene el modelo tridimensional, fue necesaria una plataforma digital y para el desarrollo de esta, fue necesario determinar el lenguaje de programación que se utilizaría para todo el proyecto. Para la selección del lenguaje de programación se realizó una investigación preliminar acerca de lenguajes que tuvieran la capacidad de desplegar gráficas tridimensionales, detección de eventos como teclado o ratón y la lectura de archivos en formato STL.

Para poder modificar el modelo fue necesario la lectura de la información contenida en el archivo, y para esto se implementó un algoritmo para que lea y guarde la información del documento con formato STL a un arreglo de datos en el programa y viceversa.

Con la información cargada en el programa, fue necesario diseñar un algoritmo para enviar los datos de forma correcta al renderizador. El algoritmo utiliza el vector normal de cada una de las partes del modelo para identificar si los vértices se encuentran en el orden correcto y de no estarlo se invierte, finalmente envía la información al renderizador para desplegar en pantalla el modelo tridimensional.

Se investigó acerca de algoritmos para el corte de una malla de puntos de un objeto tridimensional y se implementó una variación de uno de estos algoritmos por medio de un plano para representar la sección de corte y finalmente se agregó la funcionalidad de exportación del modelo digital modificado a un archivo con formato STL.

Abstract

This work presents the process of modification of a three dimensional model. This is one of three necessary processes to get a medical heart exam to a physical model, these are: the transformation of a medical exam to a digital 3D model, modification and printing of the 3D model.

To perform the modification and generation of the file containing the three dimensional model, a digital platform was necessary and for the development of the platform it was necessary to select a programming language that will be used for the rest of the project. To select the programming language a preliminary research about languages that had the ability to display 3D graphics, mouse and keyboard events detection and STL files reading capability.

Before starting the modification of the model, it was necessary to read the information stored in the file, to do this an algorithm was implemented to read and store the data of the STL file to an array in the program and vice versa.

With the information loaded, another algorithm was necessary to send the data in the correct form to the render. The algorithm uses the normal vector of each of the model parts to identify if the vertex is in the correct order and if is not, invert it. Finally send the information to the render to display the model on the screen.

A research about mesh cutting algorithms was performed and a variation of one of this algorithms was implemented using a plane to represent the cutting section. Finally the functionality to export the modified 3D model to a STL file was added.

CAPÍTULO 1

Introducción

Las operaciones del corazón son procedimientos que llevan practicándose durante varios años en el campo de la medicina. Una operación puede durar de tres a seis horas [1], dependiendo del procedimiento y los imprevistos que se encuentren durante la operación. Algunos imprevistos pueden ser malformaciones que no se identificaron en los exámenes, los dispositivos que se van a utilizar durante la operación tienen un tamaño inadecuado para el paciente, entre otros. Varios imprevistos podrían evitarse si los médicos tuvieran acceso a un modelo físico del órgano del paciente; este recurso se podría obtener imprimiendo un modelo del órgano del paciente utilizando impresoras precisas, como las impresoras de tipo SLA que cuentan con una resolución de 0.14mm [2], además de ser un proceso de bajo costo comparado a los procesos de fabricación tradicionales como el soplado de plástico o maquinado.

Para poder llevar un examen médico a un modelo físico existen varios pasos como la transformación del examen médico a un modelo 3D, la modificación del modelo y la generación del archivo para su impresión y la propia impresión (Figura 1). Este trabajo se enfoca en el desarrollo de una plataforma para la visualización, modificación del modelo 3D y la generación del archivo para el proceso de impresión. Cuenta con la posibilidad de realizar modificaciones simples al modelo, como cortes en distintos ángulos para dar la flexibilidad de imprimir una porción del corazón en vez del modelo completo, los cortes se realizan por medio de un plano que permite visualizar la forma del corte antes de realizar la acción.

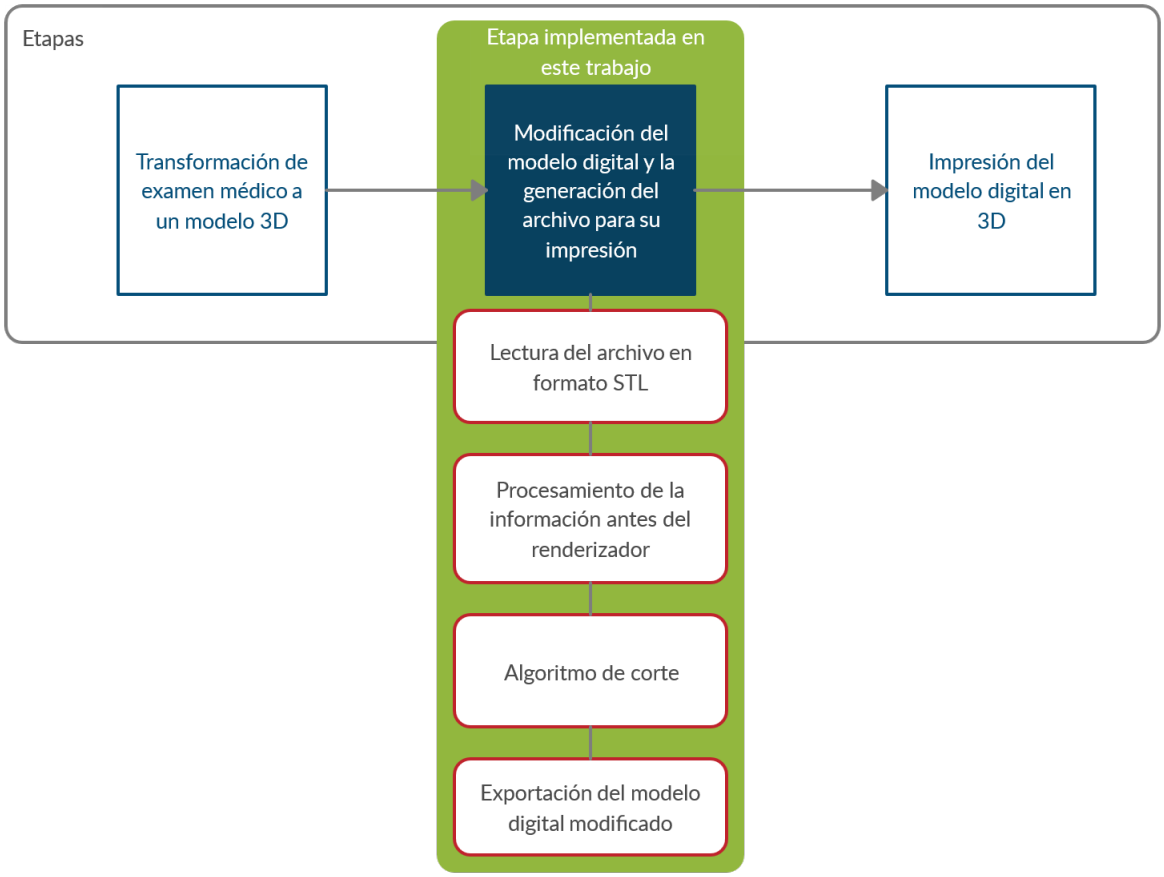

Figura 1: Diagrama del proyecto completo y la fase que se presenta en este trabajo.

Fuente: Elaboración propia

CAPÍTULO 2

Antecedentes

Un trabajo similar es 3D slicer, esta plataforma en proceso de desarrollo permite el análisis y visualización de imágenes médicas por medio de una interfaz gráfica, además posee distintas funcionalidades que pueden ser agregadas por medio de plug-ins los cuales han sido creados por distintas personas bajo un tipo de licencia de código abierto. La plataforma cumple la necesidad de tener un programa que permita las condiciones necesarias para poner a prueba algoritmos para el análisis de imágenes médicas. [3]

En el trabajo "3D Slice as an image computing plataform for the quantitative imaging network" se demuestra la utilidad de un software especializado en el análisis y procesamiento de imágenes médicas sin la necesidad de equipo especial por parte del usuario final. [4]

Ambos trabajos se enfocan en la utilidad de poseer un software que pueda procesar imágenes médicas y proporcionar funcionalidades que las contra partes comerciales no tienen implementadas. Se toma en cuenta también las ventajas de poder agregar herramientas al programa conforme se van requiriendo o dependiendo del entorno en que se utilice la plataforma.

CAPÍTULO 3

Justificación

La integración de los avances tecnológicos en la rama de la medicina ha permitido la realización de procesos cada vez más complejos o hacer los procesos existentes más seguros y en menos tiempo. Algunos procedimientos médicos ya han implementado una fase de planificación previo a la operación utilizando un modelo físico generado por medio de los datos de exámenes médicos y su edición digital. Un ejemplo es el cambio de un catéter para la válvula aorta, en el que se puso a prueba la viabilidad de realizar un análisis previo basado en el modelo tridimensional impreso [5].

Las operaciones del corazón hasta hace pocos años se analizaban por medio de escáneres que permitían observar la fisiología del órgano. Este tipo de análisis presenta la limitación de que solo se puede ver el órgano por medio de cortes transversales en una secuencia de imágenes y no poder verificar si las herramientas que se utilizarán para ejecutar la operación podrán ser utilizadas correctamente en el espacio reducido. Se realizó la planificación de una intervención estructural percutánea utilizando un modelo tridimensional impreso para verificar que los instrumentos pudieran ingresar hasta donde eran necesarios [6]. Con los avances en la precisión que una impresora 3D puede ofrecer es posible generar modelos con un gran nivel de detalle y a bajo costo.

La plataforma permitirá a los médicos observar el modelo tridimensional, interactuar con este de manera digital previo a la impresión e incluso indicar si se quiere imprimir solo una parte del objeto por medio de operaciones gráficas simples e intuitivas, este proceso permite imprimir únicamente las secciones de interés. Actualmente existen plataformas que generan los archivos para su impresión, sin embargo, estos programas no están orientados a personas con poca o ninguna práctica en modelado digital tridimensional [7].

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Desarrollar una plataforma digital que permita la modificación de modelos tridimensionales de corazones de pacientes en formato STL, su visualización durante estas modificaciones por medio de una interfaz gráfica y la generación del archivo en formato STL utilizando un algoritmo.

4.2. Objetivos específicos

- Determinar el lenguaje de programación que permita desplegar gráficas tridimensionales y compatibles con sistemas operativos Windows.
- Desarrollar una interfaz donde se desplegará el modelo tridimensional usando elementos gráficos.
- Implementar la exportación del modelo modificado a un archivo STL por medio de un algoritmo.
- Realizar pruebas con archivos en formato STL diferentes para comprobar que se exportan correctamente desde la plataforma

CAPÍTULO 5

Marco teórico

5.1. Gráficos por computadora

Bajo el contexto de la representación de objetos tridimensionales sin importar la cantidad de detalles que puedan llegar a tener, todos los objetos son representados por medio de una nube de puntos en el espacio denominados vértices, los bordes que son líneas representando la conexión entre dos vértices, y las caras que son elementos triangulares que se componen de un conjunto de vértices y bordes (Figura 2) .

Existen distintos tipos de estructuras de datos para poder almacenar la información de un objeto tridimensional y cada una es utilizada en distintas aplicaciones. Dos de los formatos más utilizados y aceptados por la mayoría de programas de modelado tridimensional son la estructura basada en triángulos y la estructura basada en índice de caras. La estructura basada en triángulos es utilizada para definir los archivos de tipo STL, en el caso de los

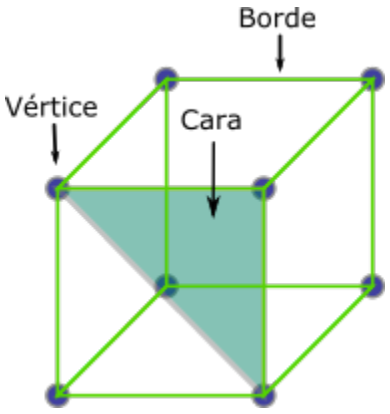

Figura 2: Elementos de un objeto tridimensional

Fuente: Elaboración propia

Figura 3: Estructura basada en triángulos sin duplicidad de datos

Fuente: Elaboración propia

Figura 4: Estructura basada en índice de caras

Fuente: Elaboración propia

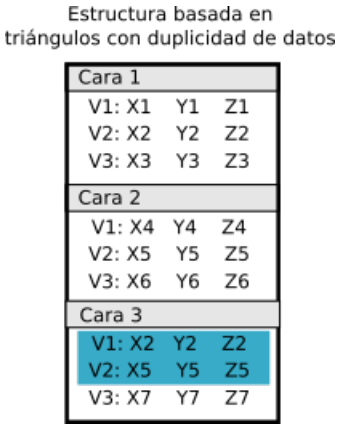

Figura 5: Estructura basada en triángulos con duplicidad de datos

Fuente: Elaboración propia

archivos con extensión OBJ utilizan una estructura basada en índice de caras.

Una estructura de datos basada en triángulos almacena la información de cada uno de los vértices de cada uno de los triángulos del objeto. Este tipo de estructura no guarda información sobre alguna conexión entre los vértices (Figura 3), esto significa dependiendo de la figura que se represente puede existir duplicidad de datos y en el peor de los casos la información puede estar duplicada muchas veces porque varias caras comparten un mismo vértice, haciendo que los archivos sean muy grandes. Figura 5.

La estructura basada en un índice de caras se forma utilizando una lista en la que se encuentra la información de todos los vértices utilizados por el objeto y otra lista que contiene la información de las caras, representadas por índices que corresponden al arreglo de vértices [8] (Figura 4). Este tipo de arreglo puede contener información acerca de vectores normales de cada vértice, esta información es importante para determinar como se comportará la luz en toda la superficie del objeto [9]. Otro componente que se puede agregar a esta estructura es la información de la textura que se utilizará en el modelo de ser necesaria.

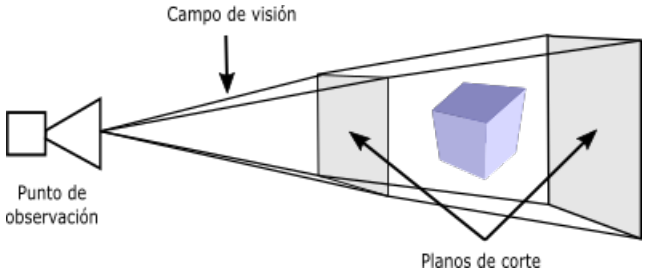

Figura 6: Estructura utilizada para el dibujado de un objeto tridimensional

Fuente: Elaboración propia

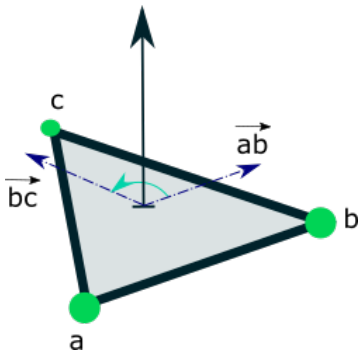

Figura 7: Cara con la normal en dirección de las agujas del reloj.

Fuente: Elaboración propia

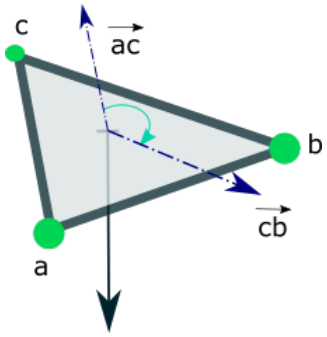

Figura 8: Cara con la normal en dirección en contra de las agujas del reloj.

Fuente: Elaboración propia

5.1.1. Dibujado de gráficos

Con el paso del tiempo la capacidad computacional a aumentado permitiendo que los modelos sean cada vez más complejos, sin embargo cuando se trata de procesar un modelo con gran cantidad de detalles en tiempo real es necesario hacer el proceso de dibujado de forma eficiente, la mayoría de programas realiza este proceso dibujando únicamente las superficies visibles en un espacio determinado. Este espacio es definido utilizando planos de corte con dimensiones ancho y largo específicos que indican la distancia mínima como máxima de donde el objeto puede dibujarse, y un campo de visión que se define proyectando vectores desde un punto de observación hasta las esquinas de los planos de corte (Figura 6). Adicional a esto se analiza cada cara del objeto para determinar si es necesario dibujarla, este análisis se realiza evaluando la dirección de cada una de las caras con respecto al punto de observación. Si la normal de la cara está dirigida en dirección al punto de observación esta se dibuja, en caso contrario se omite el envío de información a la pantalla. Esto previene el procesado de información que no será visible al usuario.

Todas las caras de un objeto poseen una dirección normal que se puede calcular realizando el producto cruz entre los vectores generados por los dos primeros bordes de la cara. Tomando como ejemplo un triángulo con vértices a, b, c; si el triángulo se dibuja en el orden $(\vec{ab}, \vec{bc}, \vec{ca})$ se dice que la cara está orientada en dirección en contra de las agujas del reloj (Figura 7). Si un triángulo definido por los mismos vértices se dibuja pero en el orden $(\vec{ac}, \vec{cb}, \vec{ba})$ se

dice que está orientado en dirección de las agujas del reloj (Figura 8) [10]. En la mayoría de programas encargados de dibujar objetos tridimensionales se puede indicar si se dibujan solo las caras orientadas en dirección horaria, antihoraria o ambas. Esta última opción no es común ya que el análisis de dirección se realiza para poder hacer eficiente el proceso de dibujado.

5.2. Archivos STL

STL es un acrónimo para *Standard Tessellation Language*, el formato de archivo STL fue desarrollado por el grupo de consultoría Albert-Battaglin Consulting Group para la compañía 3D Systems Inc. para poder representar un modelo tridimensional, generado con un programa de diseño asistido por computadora, y ser utilizado por aparatos de impresión de tipo SLA. Este formato de archivos representa la superficie de objetos tridimensionales por medio de triángulos unidos entre sí [11]. Actualmente los archivos STL se utilizan para impresión 3D, algunos programas de manufactura asistida por computadora requieren este formato para realizar los cálculos y como medio de comunicación de información entre programas CAD y CAM.

Este tipo de archivo almacena la información utilizando la estructura de datos basada en triángulos mencionada anteriormente. Adicional a la estructura existe el formato STL ASCII, que representa la información con caracteres. La única ventaja que representa es poder analizar el archivo directamente ya que al almacenar información de modelos complejos el tamaño de archivo aumenta considerablemente. Existe también el formato STL binario y este representa la información en lenguaje de bajo nivel que no es legible para el usuario pero el tamaño de los archivos es menor comparado al formato ASCII, por esta razón la mayoría de programas de edición 3D generan archivos de tipo STL binario. [12]

5.3. Multithread

Un proceso es una unidad de trabajo y los sistemas operativos modernos manejan varios procesos asignados a un solo hilo de control. Por lo regular cuando se ejecuta un programa éste está compuesto por varios procesos que se ejecutan de forma concurrente o paralela. En el caso de ejecución concurrente uno o varios programas esperan a ser ejecutados por algún núcleo del procesador una tarea a la vez, se ejecutan durante un intervalo predefinido y vuelven a esperar a ser ejecutados de nuevo. Para la ejecución paralela un programa posee varias tareas y estas se encuentran repartidas entre los distintos núcleos del procesador por lo que el programa puede obtener resultados de las distintas tareas en distinto orden al que fueron creadas.

En el caso de procesos que se ejecutan de forma paralela, estos poseen varios hilos de control que se ejecutan al mismo tiempo en núcleos distintos del procesador. Este tipo de ejecución permite a los programas realizar distintas tareas y presentar una mejor respuesta ante las acciones del usuario, esta característica es la que permite realizar operaciones intensivas sin bloquear por completo el programa [13].

A pesar de las ventajas que presenta la programación de procesos ejecutándose en paralelo existen algunos problemas que deben ser tomados en cuenta. La dependencia de información es un problema a la hora de programar de forma paralela ya que se debe sincronizar ambas tareas y así evitar el uso de información errónea o desactualizada entre las tareas. Un ejemplo de este problema puede ser un programa que realiza cálculos de información, contenida en una lista, en un proceso y el despliegue de la información en pantalla en otro proceso; si se despliega la lista de información en pantalla en el mismo instante en el que se almacena un cálculo, no se puede asegurar que se despliega lo último que se ha escrito o que la información esté incompleta. Una solución para la sincronización de tareas es utilizando *Mutex Locks* y consiste en bloquear el acceso a la información para casi todas las tareas excepto para la tarea que haya solicitado el bloqueo previo a la ejecución de ciertas instrucciones y al terminar de utilizar la información esta queda disponible de nuevo para el resto de procesos. Normalmente se utiliza un identificador que se comparte entre los procesos para saber si la información está disponible o está siendo utilizada.

Este método no asegura que dos tareas traten de bloquear la información al mismo tiempo por lo que se debe de tomar en cuenta a la hora de implementarlo, sin embargo existen situaciones en las que el método es suficiente para la sincronización entre procesos.

5.4. Unity - Plataforma de desarrollo en tiempo real

Unity fue lanzado en 2005 como un motor de videojuegos. Uno de los creadores explicó que un motor de videojuegos es un conjunto de herramientas que se utilizan para la creación de juegos, que facilita el uso de gráficas, audio, físicas, interacciones y redes. Todas estas herramientas deben de ser eficientes ya que se trabaja con gran cantidad de información en aplicaciones como los videojuegos [14]. Unity ha sido utilizado principalmente para la creación de juegos, sin embargo ha sido utilizado para el renderizado de escenas realistas para presentación de proyectos, herramientas para edición CAD o para desarrollar aplicaciones para dar servicio a clientes.

Unity posee distintas herramientas que permiten rápida edición e iteración durante el ciclo de desarrollo además de poseer previsualización en tiempo real. Posee además soporte para el desarrollo de aplicaciones 2D como 3D, herramientas para la creación de interfaz de usuario y la posibilidad de implementar código personalizado basado en el lenguaje C# [15]. La posibilidad de agregar código personalizado permite agregar funcionalidades que la plataforma no tiene como la lectura de archivos con ciertos formatos como STL, o la modificación de los objetos de los que dispone la plataforma para que tengan un comportamiento distinto como por ejemplo el renderizado de información que un algoritmo le envíe.

Una escena en Unity está compuesta por entidades y estas a su vez poseen componentes que definen el comportamiento que tendrá cada una de las entidades [16]. Por ejemplo la entidad que funcionará como cámara posee un componente de tipo cámara, el cual posee los parámetros necesarios para definir el punto de observación, los planos de corte y el campo de visión entre otras características.

Además de poder agregar componentes predefinidos, Unity brinda la funcionalidad de un componente de tipo código en el que se puede programar el comportamiento de los objetos,

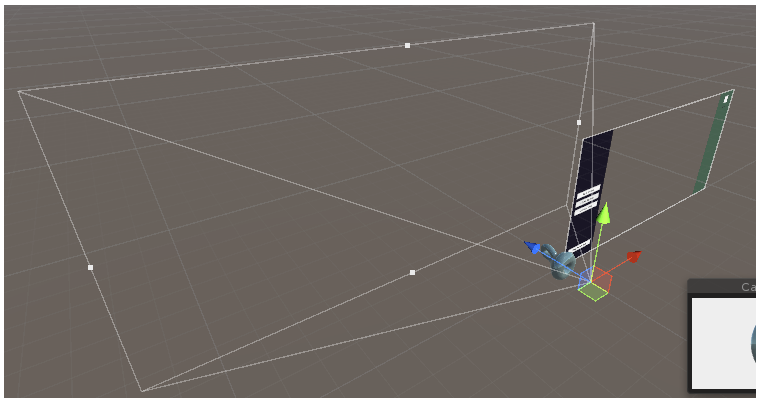

Figura 9: Elementos de una escena en unity

Fuente: Elaboración propia

cómo interactúan entre ellos, he incluso obtener información sobre otros objetos.

Unity utiliza un hilo de ejecución principal para actualizar todos los objetos en la escena [17], este hilo se puede separar en varias etapas las cuales son:

- Inicialización
- Editor
- Proceso previo al primer cuadro
- Actualización del cuadro
  - Físicas
  - Eventos de entrada
  - Lógica de juego
  - Dibujado de escena
  - Dibujado de Gizmos
  - Dibujado de interfaz gráfica
  - Fin del cuadro
  - Pausa
- Descomposición

5.5. Puntos, vectores y planos en el espacio tridimensional

Un punto en el espacio se puede definir por medio de sus coordenadas x,y,z. Un vector es un segmento de recta dirigido que corresponde a un desplazamiento desde un punto A hasta un punto B, y el vector de A a B se denota como $\vec{AB}$ [18]. Con estos dos elementos es posible definir una recta que en su forma vectorial está conformada por un punto que existe sobre la recta y un vector director (d), que es un vector paralelo a la recta:

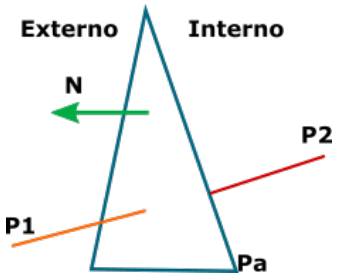

Figura 10: Representación de la escena de dos puntos pertenecientes a un segmento de recta atravesando un plano

Fuente: Elaboración propia

$$x = p + \mathbf{t}d \tag{1}$$

En el caso de un plano es posible definir su forma vectorial por medio de dos vectores directores (u, v) y un punto que pertenezca al plano:

$$x = p_p + d_b\mathbf{u} + d_c\mathbf{v} \tag{2}$$

Existe un escenario en el que se necesita identificar si un punto se encuentra en uno u otro lado de un plano. Primero es necesario identificar los lados del plano, estos lados se definirán como externo al lado en dirección a la normal del plano e interno al lado en dirección opuesta a la normal, el escenario completo se puede ver en la Figura 10. La selección de lados se realiza de esta manera para mantener la consistencia con respecto a las normales de los planos tridimensionales [19].

Utilizando la definición del producto punto entre vectores:

$$\vec{U} \bullet \vec{V} = ||\vec{U}||||\vec{V}|| * cos(\theta) \tag{3}$$

y tomando en cuenta que la norma de un vector siempre será positiva dado que este valor es la raíz cuadrada de la suma de sus componentes al cuadro, es decir:

$$||\vec{V}|| = \sqrt{{v_1}^2 + {v_2}^2 + ... + {v_n}^2} \tag{4}$$

Se pueden hacer las siguientes afirmaciones:

- Si el ángulo entre dos vectores es agudo, entonces el resultado del producto punto entre los dos vectores es positivo.
- Si el ángulo entre dos vectores es recto (son perpendiculares), el resultado del producto punto entre estos vectores será cero.

- Si el ángulo entre dos vectores es obtuso, entonces el resultado del producto punto entre los dos vectores será negativo.

Con esto es posible determinar de que lado del plano se encuentra un punto en el espacio (Pt), utilizando un punto perteneciente al plano (Pa) y la normal del plano (N). El cálculo se realiza de la siguiente manera:

$$Lado = N \bullet (P_t - P_a) \tag{5}$$

Otra situación importante es cuando un segmento de recta cruza un plano y estamos interesados en determinar la posición del punto que se genera cuando los dos elementos se cruzan. Para resolver este problema es necesario resolver el sistema de ecuaciones lineales que resulta de igualar las ecuaciones vectoriales de una linea y un plano:

$$p + d\mathbf{t} = p_p + d_b\mathbf{u} + d_c\mathbf{v} \tag{6}$$

Resolviendo el sistema de ecuaciones lineales se puede obtener el siguiente resultado para el parámetro t:

$$t = -\frac{(d_b \times d_c) \bullet (p - p_p)}{d \bullet (d_b \times d_c)} \tag{7}$$

sustituyendo este valor de t en la ecuación de la recta que cruza el plano se pueden obtener las coordenadas del punto donde la recta cruza al plano:

$$P_{cruce} = p - d * \frac{(d_b \times d_c) \bullet (p - p_p)}{d \bullet (d_b \times d_c)} \tag{8}$$

5.6. Algoritmos de corte

El corte de una malla de puntos se puede reducir a varios pasos los cuales dependiendo del método que se implemente pueden variar. El primer método se presenta en el trabajo [20] en el que proponen los siguientes pasos para el corte de un objeto tridimensional:

- Definir la malla de polígonos y el plano que se utilizará para el corte
- Cortar cada una de las caras que intersectan el plano y dividir la malla en dos partes separadas
- Construir la geometría interna revelada por el corte.

En el paso de identificar la malla de polígonos se asume que la información se encuentra almacenada por medio de un índice de caras, por lo que asumen que un vértice no se repite.

Para el corte de los polígonos se utiliza la relación que existe entre cada uno de los vértices para generar los vectores que representan los bordes del triángulo, los cuales son evaluados para identificar si el borde y el plano se intersectan y las coordenadas del punto de intersección, estas coordenadas pueden encontrarse con el método mencionado en la sección anterior. Además presenta tres posibles casos que pueden ocurrir durante el corte de una malla de polígonos, para los primeros dos casos el triángulo no se intersecta con el plano de corte ya que se encuentra por completo en uno de los lados. El tercer caso se da cuando el plano y el polígono intersectan, en este caso se utiliza una ecuación para interpolar linealmente la posición de los dos nuevos vértices necesarios para realizar el corte del triángulo.

En este trabajo solo se toma en cuenta el corte para objetos sin geometría interna para hacer la construcción de una cara luego del corte.

Un segundo método es presentado en el trabajo [21] en el que se utiliza otro tipo de análisis previo a realizar el corte, este consiste en los siguientes pasos:

- Mover los vértices del modelo hacia un vértice sobre la línea de corte solo si la distancia entre ellos es menor o igual a un valor permitido
- Se busca el vértice del modelo más cercano a la linea de corte y se mueve la línea de corte hacia el vértice.
- Simplificar la linea de corte, dejando únicamente vértices de esta linea que intersecten con los bordes del polígono.
- Asegurar que un triángulo es cortado en no más de dos piezas.

El segundo método presentado propone modificaciones a la malla de polígonos y al objeto de corte para realizar menos operaciones. Para el primer método la simplificación es asegurar que un triángulo es cortado por medio de un plano. Ambos métodos pueden ser aplicados para la edición en tiempo real ya que no requieren de muchas operaciones por polígono para realizar el corte.

CAPÍTULO 6

Selección del lenguaje de programación

Previo al desarrollo de la plataforma fue necesaria la selección de las herramientas de software que se utilizarían para el proyecto. Se utilizaron los siguientes criterios para identificar la mejor opción:

- **Manejo de gráficos 3D** Ya que la plataforma está enfocada en desplegar modelos tridimensionales digitales es necesario que la herramienta de software posea esta funcionalidad.
- **Soporte para la creación de interfaces gráficas** Uno de los objetivos es desplegar los modelos tridimensionales por medio de una interfaz gráfica, por lo que es necesario que la herramienta posea un método para la creación de interfaces gráficas.
- **Detección de eventos de teclado y ratón** Ya que el usuario estará interactuando con la plataforma directamente, es necesaria la detección de ingreso por medio de teclado o acciones con el ratón.
- **Poca dependencia de librerías** Al ser un proyecto que continuará en desarrollo la dependencia de librerías puede representar un problema si una de estas deja de tener soporte para las nuevas versiones del software seleccionado. Por lo que se busca que la herramienta de software no dependa principalmente de librerías para cumplir con los criterios anteriores.

6.1. Prueba básica para comprobar los criterios de selección

Para comprobar los criterios de selección se realizó una prueba que consistió en:

- Dibujar un cubo en pantalla con valores agregados en código

- Creación de un botón y la impresión de un mensaje de prueba cuando éste es presionado
- Impresión de un mensaje cuando una tecla del teclado es presionada y la impresión de la posición del ratón

6.2. Pruebas realizadas

6.2.1. Python

Este lenguaje de programación no posee manejo de gráficos tridimensionales de forma nativa así como el soporte para la creación de interfaces gráficas y la detección de eventos de teclado y ratón por lo que fue necesario el uso de librerías para cumplir con los tres primeros criterios. Esto excluye el último ya que fueron necesarias librerías para cumplir con los criterios anteriores.

6.2.2. Java

Este lenguaje de programación posee soporte para el manejo de gráficos 3D por medio de librerías como JavaFx. Java posee soporte nativo para la creación de interfaces gráficas y detección de eventos de teclado y ratón. Se puede decir que cumple con el criterio de poca dependencia de librerías ya que únicamente fue necesario el uso de una librería.

6.2.3. Unity

Esta opción no es un lenguaje de programación sino que es una plataforma para desarrollo de aplicaciones en tiempo real que permite la utilización de código en lenguaje C#. Esta plataforma posee de forma nativa el manejo de gráficos 2D como 3D, soporte para la creación de interfaces gráficas y detección de eventos de teclado y ratón sin necesidad de librerías.

Se seleccionó la plataforma de desarrollo Unity junto con el lenguaje C# como las herramientas para la creación del proyecto ya que cumplió con los criterios sin necesitar de librerías.

CAPÍTULO 7

Lectura del archivo en formato STL

Tomando en cuenta la existencia de dos tipos de archivos STL (binarios y ASCII), se optó por utilizar una librería escrita para C# llamada IxMilia.STL. Esta librería contemplaba ambos formatos y no era necesaria la intervención por parte del usuario o del código adicional para identificar el tipo de archivo que se había seleccionado además de cargar correctamente la información. Sin embargo a pesar de generar correctamente la información fue necesario transformarla a variables que la plataforma de Unity utiliza para realizar el cálculo vectorial.

La estructura que utiliza la librería consiste en objetos *StlTriangle, StlVertex y StlNormal*. Los objetos *StlVertex y StlNormal* poseen variables de punto flotante en la que se almacena la información de cada punto en el espacio. Unity utiliza objetos de tipo *Vector3* que permite realiar operaciones vectoriales como el cálculo de distancia entre vectores, producto cruz entre otras operaciones que fueron necesarias en otros procesos del proyecto. Tomando en cuenta las ventajas de tener la información almacenada en variables de tipo *Vector3* se creó un objeto denominado *vectorTriangle* el cual posee cuatro variables de tipo *Vector3* que corresponden a los tres tres vértices y el vector normal (Figura 11).

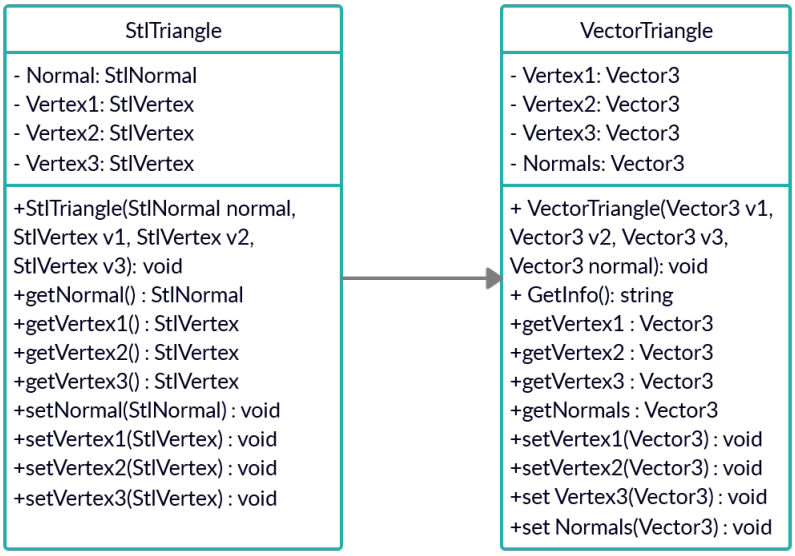

Figura 11: Conversión de objetos de tipo *IxMilia* a *vectorTriangle*

Fuente: Elaboración propia

CAPÍTULO 8

Procesamiento de la información antes del dibujado

Previo a enviar la información al renderizador fue necesario generar una lista de vértices, una lista de normales para cada uno de los vértices y una lista de triángulos que poseían índices de la lista de vértices. La lista de triángulos además contiene de forma implícita la dirección en la que se creará la cara (Figura 12).

Para asegurar que el arreglo de datos se genera correctamente se analiza cada punto previo a agregar la información. El análisis consiste en generar dos vectores para formar los primeros dos bordes del triángulo, realizar el producto cruz entre los vectores, normalizar tanto la normal proporcionada por el archivo como el vector generado por el producto cruz, y comparar la magnitud de la diferencia entre el vector normal y el obtenido para saber si se encuentran en la misma dirección o están invertidos.

Tomando como ejemplo la Figura 12, se pueden generar los siguientes vectores para iniciar el análisis:

$$\vec{V}_{AC} = (C - A)$$

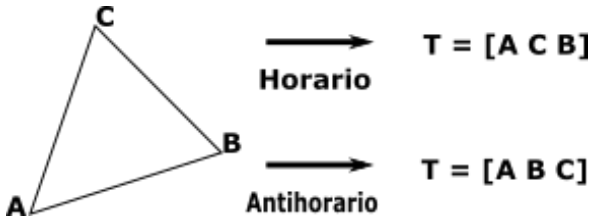

Figura 12: Arreglo de vértices para las diferentes direcciones de la cara del triángulo

Fuente: Elaboración propia

$$\vec{V}_{CB} = (B - C)$$

Se realiza el producto cruz entre los vectores $\vec{V}_{AC}$ y $\vec{V}_{CB}$ para obtener un vector normal n_a y se normalizan los vectores n_a y n_o (siendo n_o el vector normal almacenado en el archivo STL). Se realiza la diferencia entre los vectores n_a y n_o y se calcula la magnitud del vector obtenido de la diferencia entre las normales.

Si los vectores normales se encuentran en la misma dirección la magnitud será cero de lo contrario devolverá un valor distinto de cero. Con este resultado se puede saber si los vértices se almacenan correctamente, en caso contrario se invierte el orden.

CAPÍTULO 9

Algoritmo de corte

Para el proceso de corte se decidió evaluar la información por caras en vez de por puntos, esto debido a que los vértices ya se encuentran relacionados por medio de las caras y de no realizar el análisis por caras sería necesaria una reconstrucción de cada una luego de realizar el corte incurriendo en mas tiempo de procesamiento.

Para realizar el corte se optó por utilizar un plano para definir la sección de corte, para implementar el plano en la interfaz gráfica fue necesaria utilizar un objeto que contuviera dos planos con normales contrarias ya que la plataforma Unity dibuja únicamente las caras que estén definidas de forma antihoraria con respecto al punto de observación. De no hacer esto el usuario dejaba de ver el plano al pasar ciertas posiciones en la cámara.

Con el plano definido fue necesario identificar si un triángulo se encontraba del lado positivo o negativo con respecto al plano, siendo el lado positivo la dirección en la que se encuentra dirigida la normal del plano. Para poder identificar el lado en el que se encontraba se aplicó la proyección de cada uno de sus vértices, con respecto a un punto en el plano de corte, sobre el vector normal del plano de corte (Ecuación 5). Si el resultado de esta operación era positivo entonces el vértice se encontraba en el lado positivo, un resultado negativo indicaba que se encontraba en el lado negativo del plano y si el resultado era cero se tomaba como un vértice positivo si se estaba evaluando si todos los vértices se encontraban del lado positivo, y se tomaba como un vértice negativo si se estaba evaluando si todos los vértices se encontraban del lado negativo.

Con los lados de un plano definidos se pueden tener tres casos al momento de cortar un grupo de triángulos. El primer caso ocurre cuando todos los vértices del triángulo se encuentran del lado positivo del plano, el segundo es cuando todos los vértices se encuentran del lado negativo del triángulo y el tercer caso es cuando un triángulo posee un vértice en los dos lados. Dependiendo del caso se realizan las siguientes acciones:

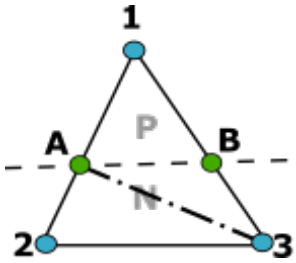

Figura 13: Representación del caso en el que un triángulo posee vértices en ambos lados del plano de corte

Fuente: Elaboración propia

Primer caso: todos los vértices se encuentran del lado positivo del plano

La información de los vértices, normal y la formación del triángulo se almacenan en un arreglo de datos donde únicamente se encuentran triángulos positivos.

Segundo caso: todos los vértices se encuentran del lado negativo del plano

La información de los vértices, normal y la formación del triángulo se almacenan en un arreglo de datos donde únicamente se encuentran triángulos negativos.

Tercer caso: un triángulo posee vértices en ambos lados del plano

En este caso siempre ocurrirá que uno de los vértices del triángulo se encuentra en el lado opuesto con respecto a los otros dos, por lo que es necesario identificarlo y para esto se aplican los siguientes pasos:

- Se realiza la proyección de cada vértice para determinar el lado en el que se encuentra cada uno con respecto al plano de corte.
- Se toma el lado de un vértice como pivote y se compara con el lado al que pertenece otro de los dos vértices restantes.
    - Si este pertenece al mismo lado el vértice restante, el que no se ha comparado, es el que se encuentra en el lado opuesto
    - Si no pertenece al mismo lado, se compara el lado al que pertenece el vértice restante y si este pertenece al mismo lado el vértice anterior es el opuesto. En caso contrario el vértice pivote es el opuesto.

Con el vértice opuesto identificado y los otros dos vértices denominados como P_2 y P_3 se pueden determinar dos nuevos vértices denominados (A y B), para determinar estos vértices se utiliza la ecuación 8.

La situación general del triángulo con vértices en ambos lados del plano y los dos nuevos vértices A y B calculados se presenta en la Figura 13.

Finalmente, tomando como ejemplo la Figura 13, se agrega un triángulo conformado por los vértices (P_1,A,B) y la normal del plano a la lista correspondiente al lado del vértice P_1.

Además se agregan dos triángulos conformados por los vértices (P_2,P_3,A) y (P_3,B,A) y la normal del plano a la lista correspondiente a los vértices P_2 y P_3.

CAPÍTULO 10

Exportación del modelo tridimensional digital modificado

Luego de que el usuario realiza las modificaciones al modelo, este puede seleccionar qué pieza modificada quiere exportar por medio de un panel que le presenta todos los objetos creados.

Para poder realizar la exportación de la información es necesaria una dirección en la que se desea guardar el archivo, para esto se le pide al usuario por medio de una ventana, asignar el nombre que tendrá el archivo y la ubicación en la que desea guardar.

Luego se procede a obtener la información contenida en el objeto de tipo *Mesh* de Unity, que se encuentra en una estructura de tipo índice de caras. La librería *IxMilia.STL* posee un método para guardar información en formato STL sin embargo, este método utiliza objetos de tipo *StlTriangle* que contiene a su vez objetos de tipo *StlNormal y StlVertex* por este motivo se transformaron los arreglos de datos del renderizador a objetos de la librería previo a la llamada del método.

CAPÍTULO 11

Implementación de la plataforma para la edición de modelos digitales 3D

11.1. Primera versión de la plataforma

Fue necesaria la creación de una primera interfaz para realizar las pruebas de selección y posterior a eso implementar los controles conformados por botones que tenían la función de llamar a los métodos con los algoritmos descritos previamente. En la Figura 14 se muestran los primeros controles y un objeto precargado, este objeto fue utilizado para realizar pruebas con los métodos que permiten el movimiento de la cámara y el plano. En la Figura 15 se muestra la interfaz con el plano de corte y los primeros controles de movimiento. En la Figura 16 se muestra la ventana donde se solicita al usuario seleccionar el archivo que desea editar, esta ventana ya tiene seleccionado por defecto el formato STL para que la pantalla muestre únicamente archivos con esta extensión.

Con esta primera interfaz fue posible poner a prueba el proceso para la lectura del archivo en formato STL descrito en el Capítulo 7. El algoritmo se ejecutaba correctamente sin embargo, al utilizar modelos más complejos por la cantidad de triángulos la plataforma dejaba de responder hasta que se finalizaba la ejecución del algoritmo. Unity posee un método llamado *update* el cual se ejecuta una vez en el hilo principal definido por Unity, utilizando este método fue posible ejecutar el algoritmo para la lectura del archivo pero el tiempo requerido para completarse aumentó. Por esta razón se decidió crear un nuevo hilo para realizar el proceso de lectura, de esta manera fue posible reducir el tiempo para el procesamiento del archivo y al mismo tiempo evitar que la plataforma deje de responder al usuario.

La Figura 17 muestra el proceso completo de lectura con la creación de un nuevo hilo.

El procesamiento de la información antes del dibujado es otro procedimiento que pudo comprobarse en esta versión de la plataforma. El algoritmo generaba la información correc-

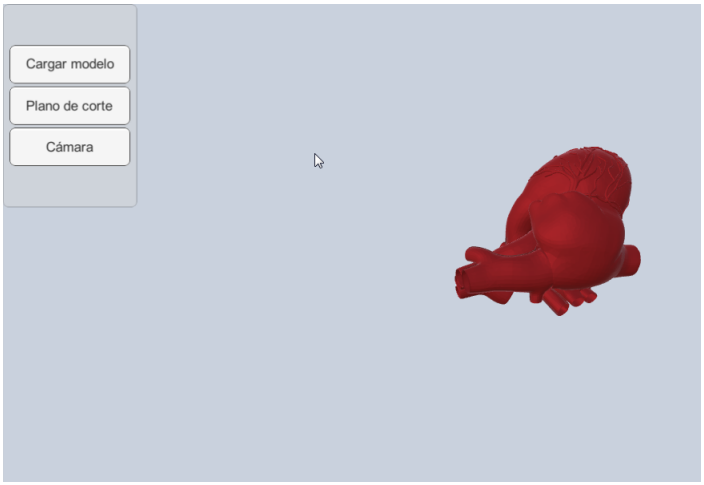

Figura 14: Primera versión de la interfaz utilizada para realizar pruebas.

Fuente: Elaboración propia

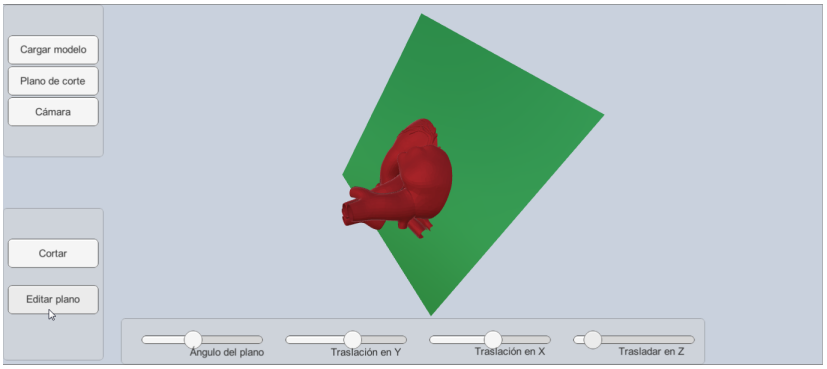

Figura 15: Primera versión de la interfaz con el plano de corte

Fuente: Elaboración propia

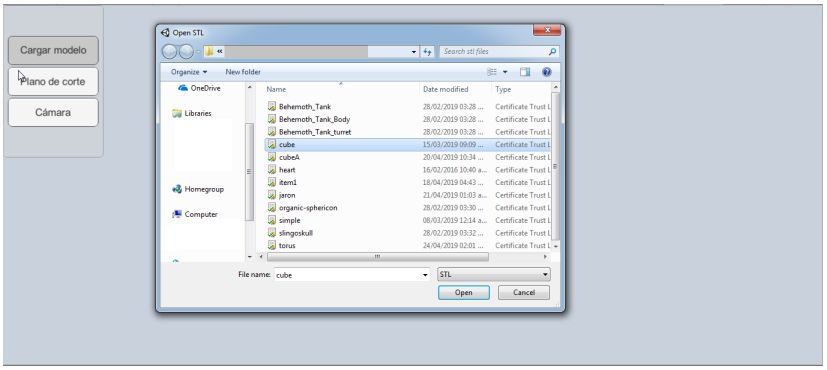

Figura 16: Primera versión de la interfaz con la ventana para seleccionar el archivo STL a editar

Fuente: Elaboración propia

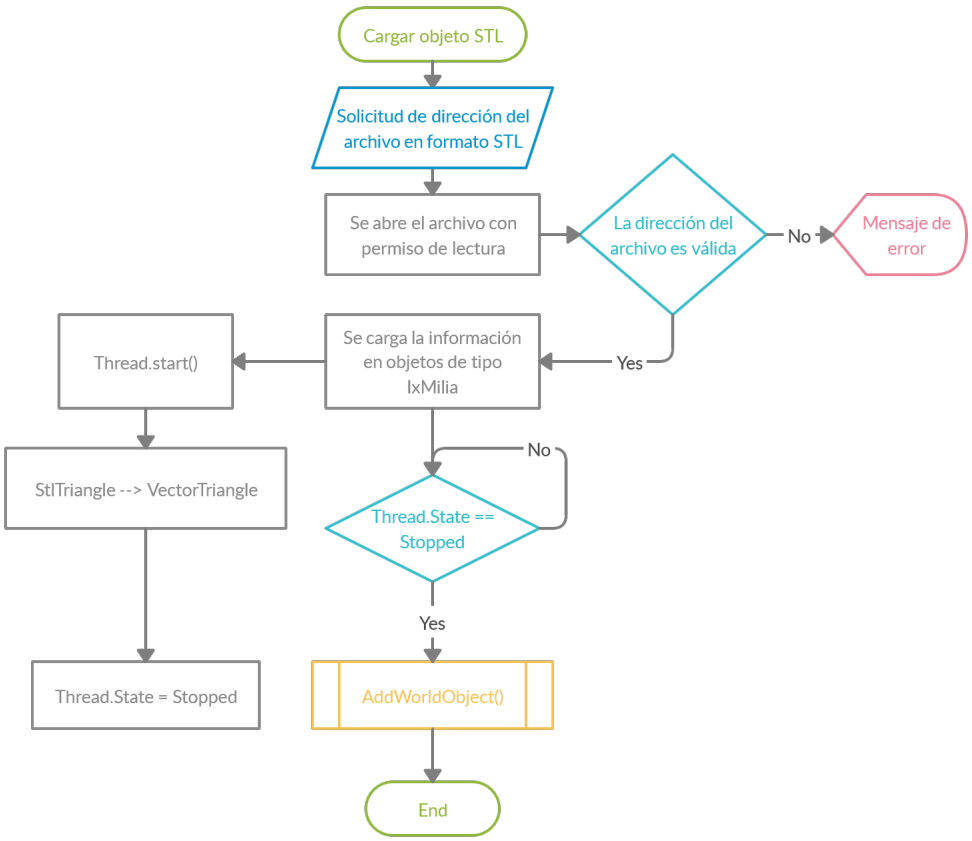

Figura 17: Diagrama de flujo del proceso de lectura del archivo con un hilo separado

Fuente: Elaboración propia

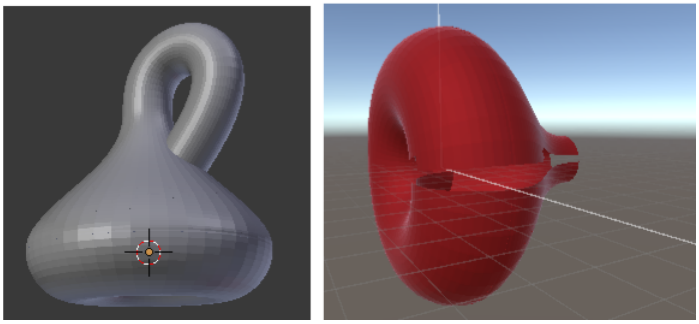

Figura 18: Botella de Klein esperada (izquierda) y botella de Klein parcialmente dibujada (derecha)

Fuente: Elaboración propia

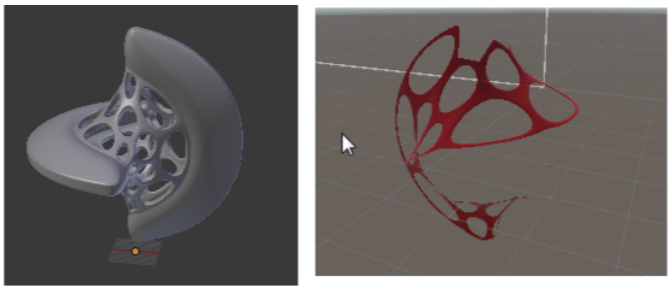

Figura 19: Esfericón esperado (izquierda) y esfericón parcialmente dibujado(derecha)

Fuente: Elaboración propia

tamente sin embargo, a la hora de dibujar el objeto éste se mostraba parcialmente. En la Figura 18 y Figura 19 se puede observar cómo se dibujan menos caras del objeto conforme aumenta la cantidad que posee. Este problema se produjo debido a que por defecto Unity utiliza índices de 16 bits, permitiendo un total de 65535 vértices, ya que ocupa menos espacio en memoria y no todas las tarjetas gráficas permiten arreglos de mayor tamaño, como en el caso de los dispositivos móviles [22]. Aún así es posible indicar el uso de índices de 32 bits para lograr arreglos con capacidad de hasta 4 billones de vértices, con esta modificación los objetos se dibujan correctamente.

11.2. Segunda versión de la plataforma

En la primera versión de la plataforma se tenía un solo objeto en escena el cual contenía los componentes necesarios para el dibujado, por lo que al utilizar el método para lectura de archivos la información se sobre escribía. Esta forma de dibujar los objetos no permitía crear nuevos de forma automática, por lo que al momento de implementar el algoritmo de corte solo se podía conservar uno de los lados. Otro de los problemas que se presentaron en la primera versión fue la falta de identificación de los objetos, esta característica se volvió necesaria para mostrar en la interfaz gráfica los objetos creados según se modificaban.

Como primer paso se creó una nueva sección en la interfaz destinada a mostrar los objetos cargados o modificados por el usuario, cada elemento de esta sección mostraría el nombre

Figura 20: Segunda versión de la interfaz con la sección destinada a mostrar los objetos cargados al lado derecho

Fuente: Elaboración propia

del objeto y una casilla de verificación para indicar si el objeto se encuentra seleccionado o no. Esta sección se muestra como un panel de color verde en el lado derecho de la pantalla del usuario (Figura 20).

La nueva interfaz implementa un arreglo de los objetos creados en el espacio. Para poder agregar un objeto al espacio de trabajo primero fue necesario implementar un algoritmo que permitiera la creación de objetos de forma dinámica, este algoritmo realiza las siguientes acciones:

- Genera un identificador para el nuevo objeto, este identificador es el valor del índice en el arreglo de objetos.
- Crea un objeto vacío (sin componentes).
- Asigna un nombre utilizando la palabra *item* y el identificador.
- Agrega un archivo con métodos que permitirán la manipulación del objeto.
- Asigna el identificador al objeto por medio de un método contenido en el archivo que se asignó previamente.
- Agrega los componentes *MeshFilter y MeshRenderer*, necesarios para dibujar información que se asigne a este objeto.
- Se agrega el nuevo objeto a la lista de objetos de la interfaz
- Se agrega una casilla de selección al panel de objetos con el nombre del nuevo objeto.

El algoritmo para agregar un objeto al espacio de trabajo se ejecutó en un hilo separado para evitar que la plataforma deje de responder a las acciones del usuario durante el procesamiento de la información. Además la verificación de la variable que indica si existen

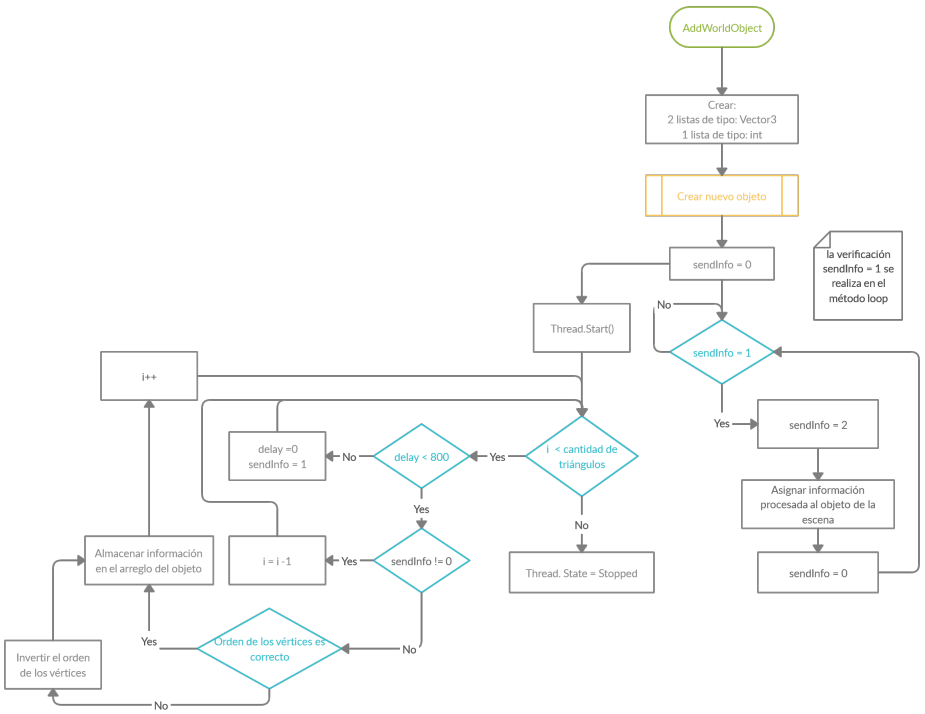

Figura 21: Diagrama de flujo para la creación de un nuevo objeto en el espacio de trabajo.

Fuente: Elaboración propia

datos que agregar al objeto, se realiza en el método Loop de Unity por lo que nunca deja de verificarse. El proceso general para la creación de un nuevo objeto en el espacio se puede observar en la Figura 21.

El siguiente paso fue la implementación del algoritmo de corte presentado en el Capítulo 9 de forma gráfica. Inicialmente se implementaron los casos en el que el triángulo pertenecía por completo a uno de los lados del plano, esto con la finalidad de verificar que el algoritmo no poseía errores de programación previo a implementar el tercer caso el cual utiliza algunos métodos de los casos anteriores.

En la Figura 22 se puede observar en el cuadro superior izquierdo que se han generado dos objetos independientes cada uno con los vértices de un lado del plano, el cuadro medio muestra el objeto cortado utilizando el plano de corte y el cuadro derecho muestra la creación de las casillas de selección de forma dinámica para cada uno de los objetos nuevos. El método de selección de objetos por medio de casillas no identificaba visualmente que objeto se tiene seleccionado, por lo que se optó por agregar un cambio de material al momento de seleccionar un objeto. El material asignado para los objetos no seleccionados consiste en un color verde no transparente (solido), y para el objeto seleccionado se asignó un material transparente azulado.

En la Figura 23 se puede observar la importancia de evaluar el caso en el que una cara posee vértices que se encuentran a ambos lados del plano de corte. El resultado es un corte no uniforme e impreciso.

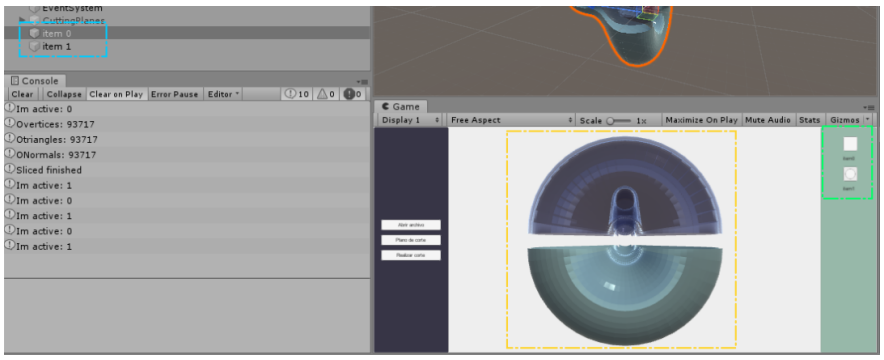

Figura 22: Verificación de la correcta creación de cada uno de los componentes luego del corte de un objeto

Fuente: Elaboración propia

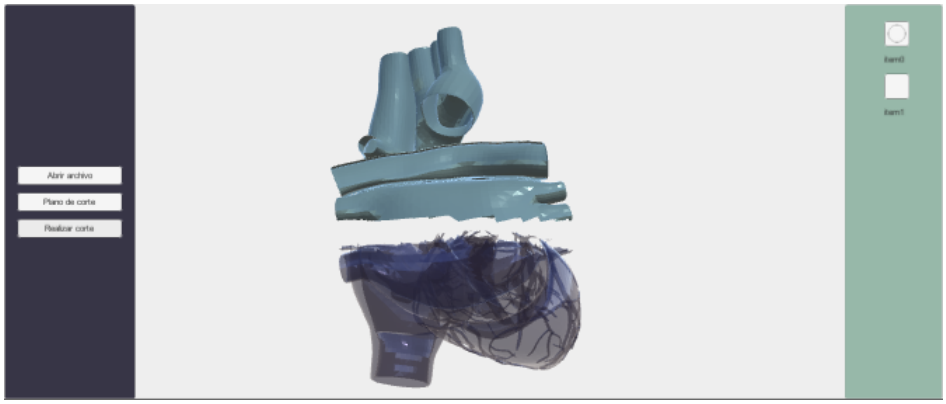

Figura 23: Resultado de un corte cuando no se toma en cuenta el caso en el que un triángulo se encuentra en ambos lados del plano.

Fuente: Elaboración propia

Figura 24: Modelo de corazón con múltiples cortes

Fuente: Elaboración propia

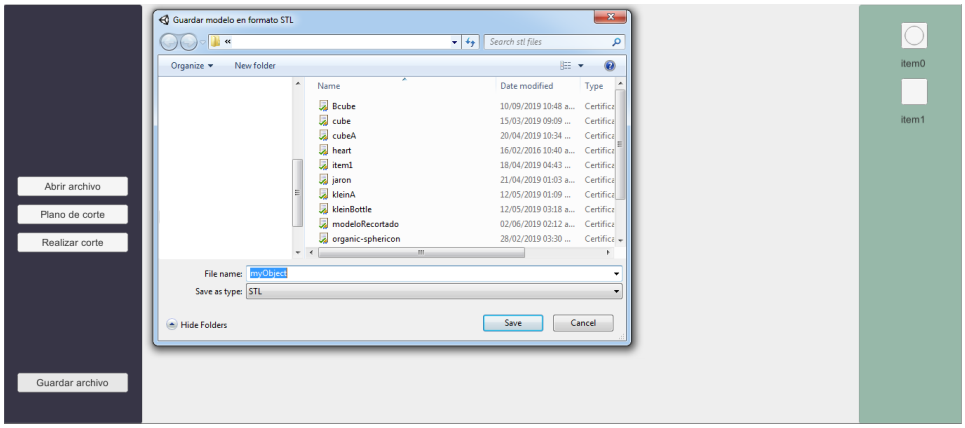

Figura 25: Ventana donde se indica la dirección donde se desea exportar el objeto modificado y el nombre.

Fuente: Elaboración propia

Como paso final se implementó el caso en el que un plano se encuentra a ambos lados del plano de corte y la implementación del método de exportación de una sección cortada. La Figura 24 muestra un modelo cortado múltiples veces. Al ser objetos separados no importa si el plano de corte atraviesa otros objetos, el análisis se realizará únicamente en las caras del objeto activo.

En la Figura 25 se puede observar la pantalla en la que se solicita al usuario indicar la ubicación donde se guardará la sección modificada y el nombre. Esta ventana provee de un nombre y el formato STL como parámetros por defecto.

CAPÍTULO 12

Resultados

Con todos los algoritmos probados se procedió a realizar pruebas con distintos tipos de archivos en formato STL, para esto se utilizaron las páginas myminifactory y thingiverse para obtener archivos con este formato. Los archivos utilizados para realizar las pruebas fueron:

- Vented and Drained Succulent Planter [23]
- Planttlement - Castle Planter with Moat, for Succulents [24]
- Anatomical Heart [25]
- Sphericon [26]

En la Figura 26 se puede observar el corte que se realizó al modelo de un corazón utilizando la plataforma, y en la Figura 27 se puede observar los diferentes ángulos las dos piezas generadas con el plano de corte (en la parte superior de la imagen) y la geometría interna que queda expuesta luego del corte para ambas piezas (en la parte inferior).

En las figuras 28 y 29 se puede observar la generación de dos piezas utilizando la plataforma, y la verificación de los dos archivos generados durante la exportación del modelo de jarrón modificado.

En la Figura 30 se muestra el corte del modelo en dos piezas utilizando la plataforma y en la Figura 31 se comprueba la correcta generación de los archivos utilizando la plataforma.

Finalmente se agregó una pantalla de inicio donde se indican los controles de la plataforma así como los iconos que se muestran cuando estos están activos (Figura 32).

Se tiene una plataforma que muestra el estado de carga del archivo en una sección de mensajes (parte central inferior), una opción para poder eliminar objetos que se encuentran

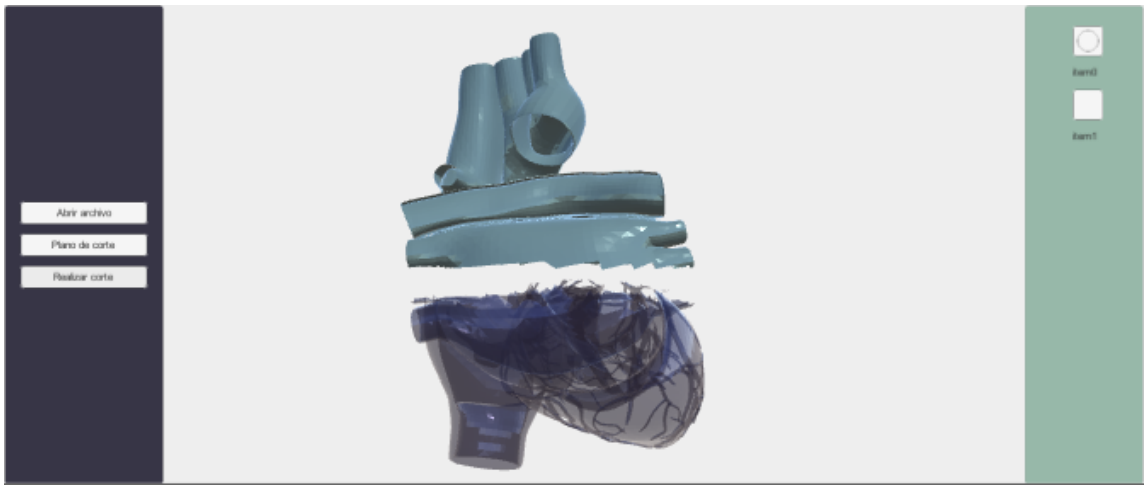

Figura 26: Prueba de corte con un modelo de un corazón

Fuente: Elaboración propia

en el espacio de trabajo, se bloquean los botones para cargar un nuevo objeto, así como el botón de corte durante la carga de archivos y un botón de salida para finalizar el programa. Todos estos cambios finales se pueden observar en la Figura 33.

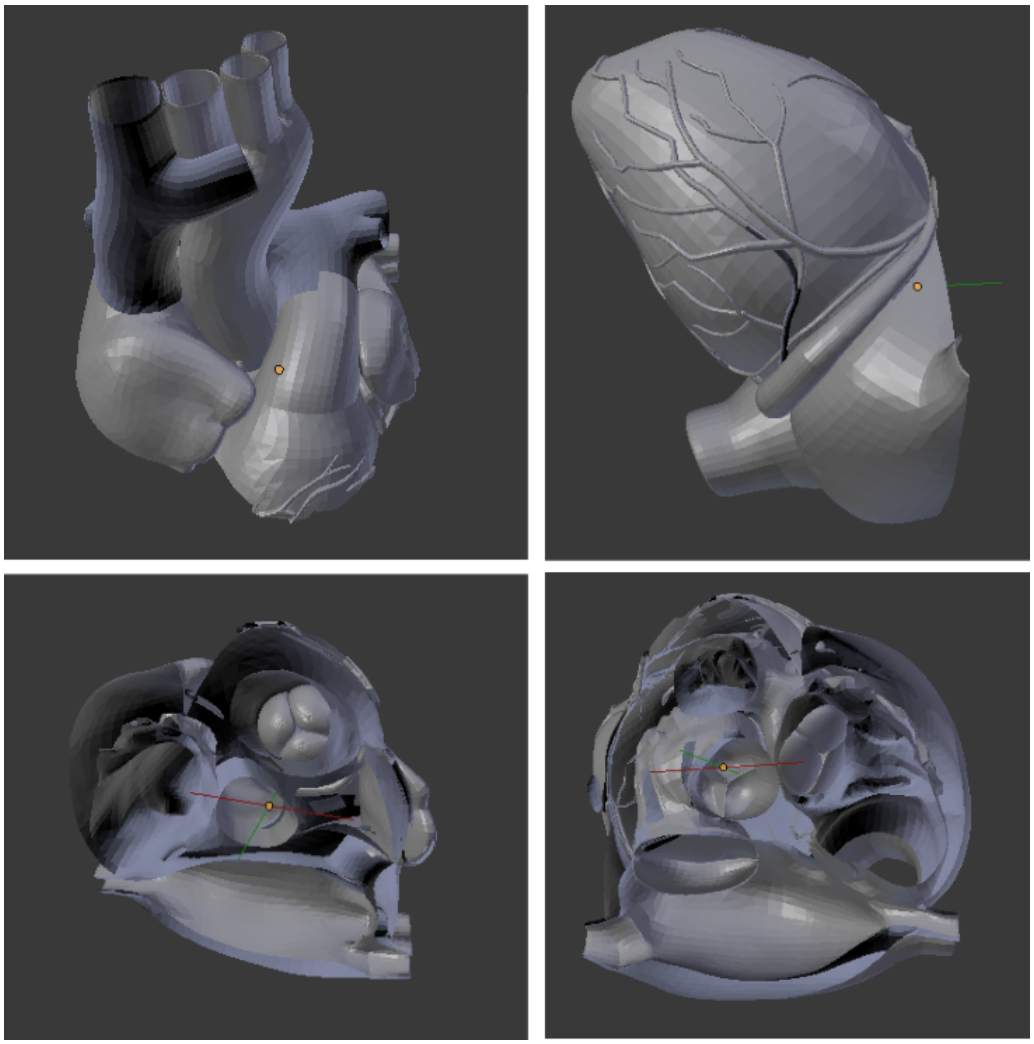

Figura 27: Verificación del corte del modelo de un corazón

Fuente: Elaboración propia

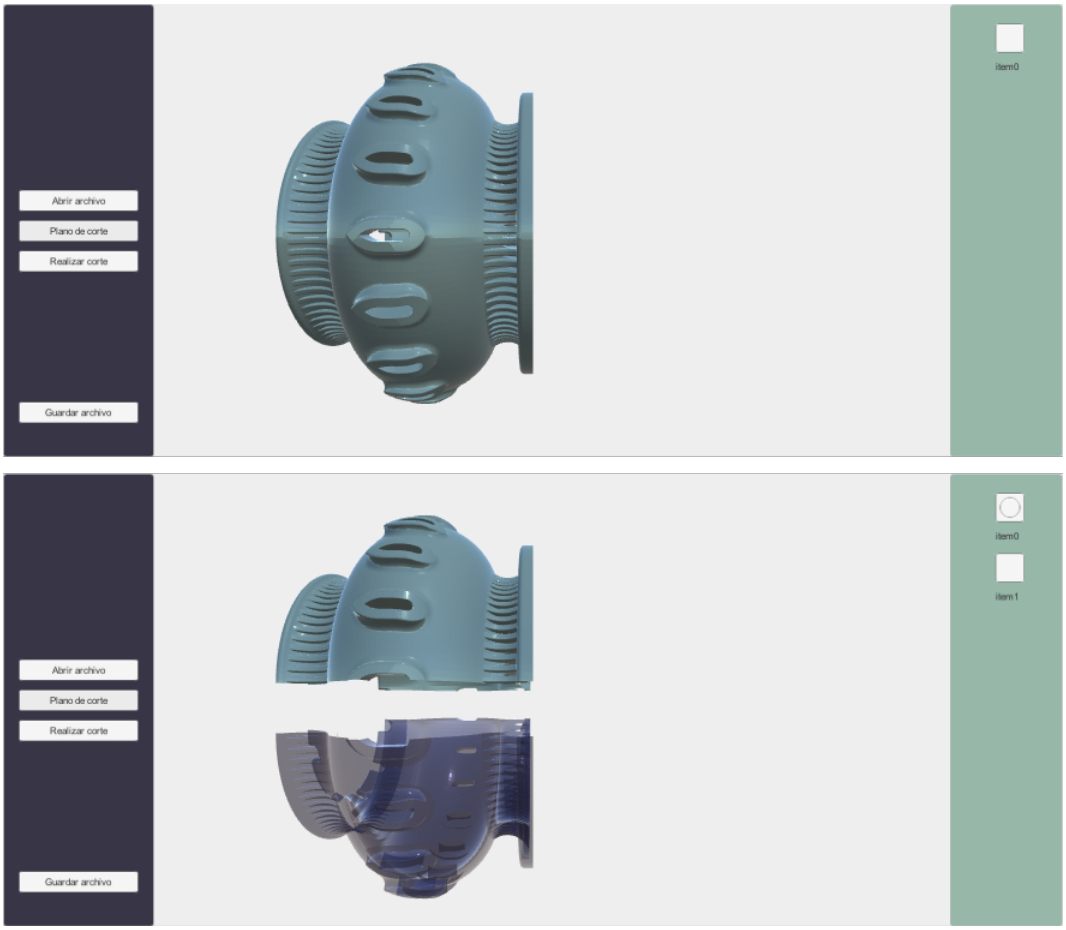

Figura 28: Prueba de corte en un modelo de jarrón tradicional

Fuente: Elaboración propia

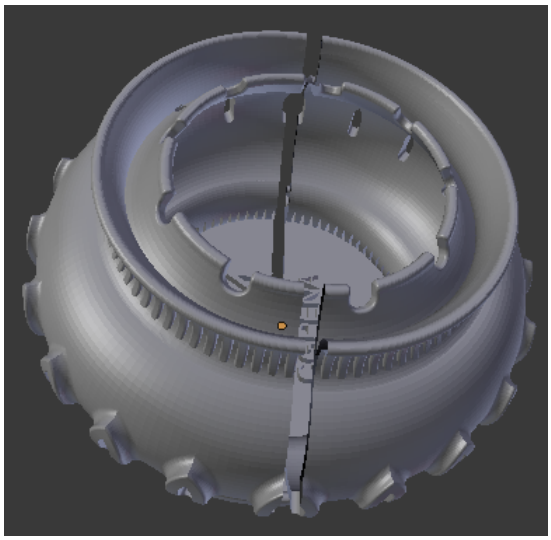

Figura 29: Verificación del corte del modelo de un jarrón tradicional

Fuente: Elaboración propia

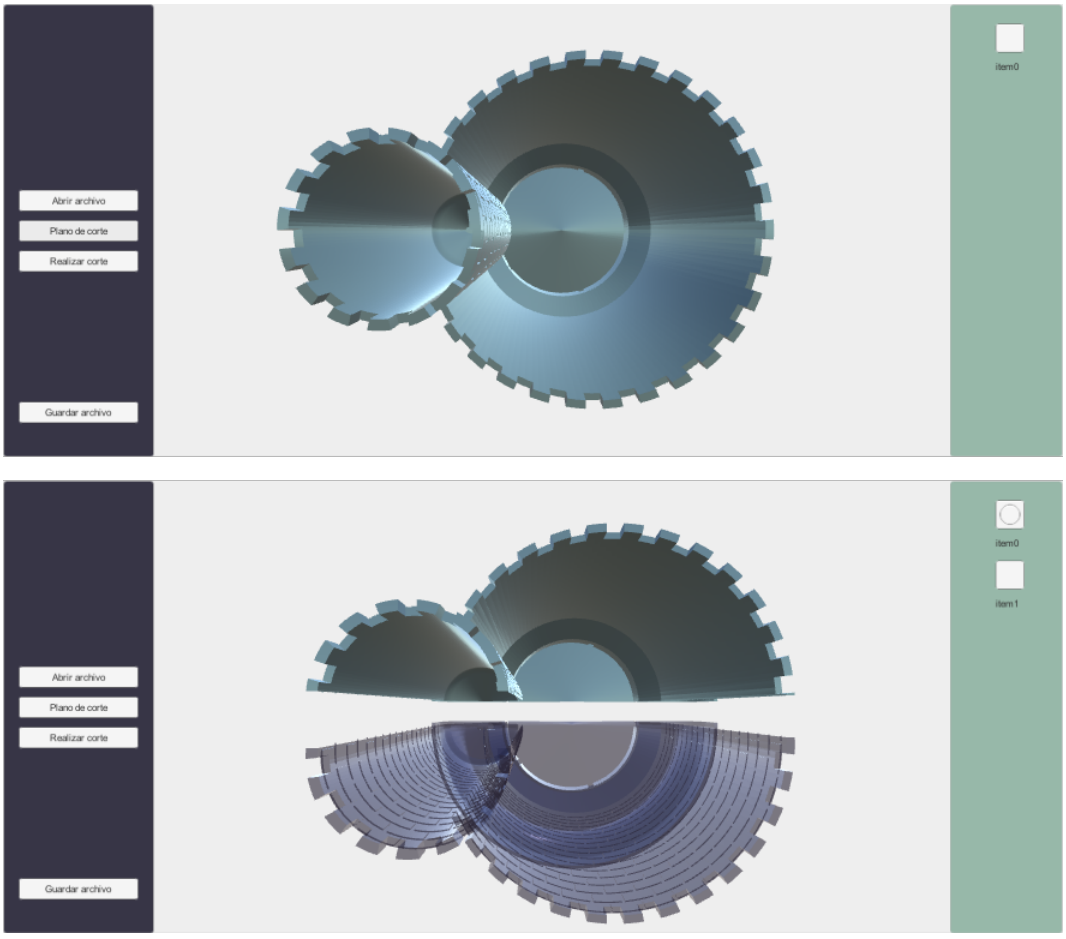

Figura 30: Prueba de corte en un modelo de jarrón con forma de castillo

Fuente: Elaboración propia

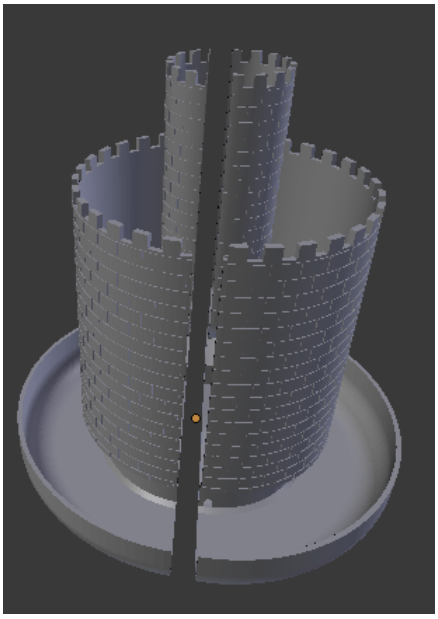

Figura 31: Verificación del corte del modelo de un jarrón con forma de castillo

Fuente: Elaboración propia

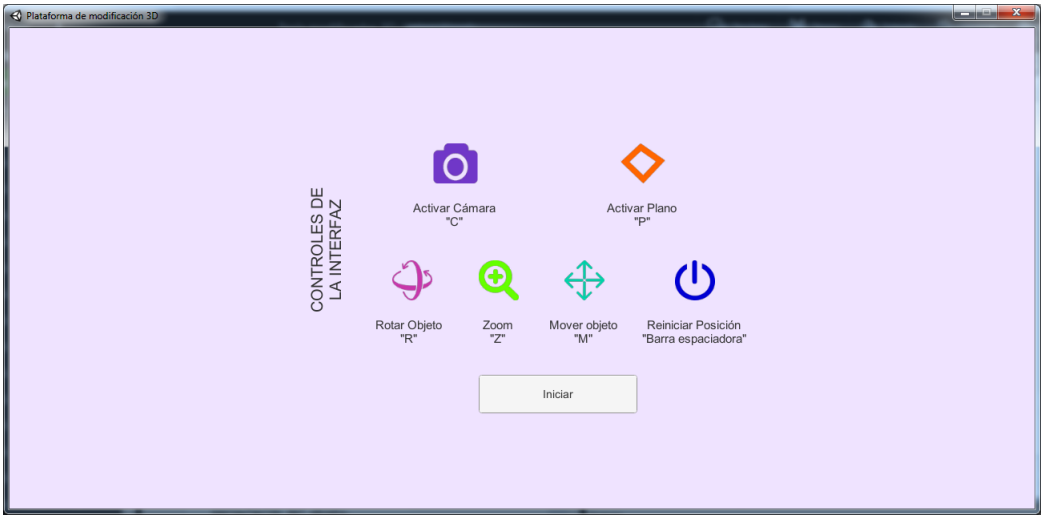

Figura 32: Primera pantalla en la que se muestran los controles así como los iconos que se presentan cuando están activos

Fuente: Elaboración propia

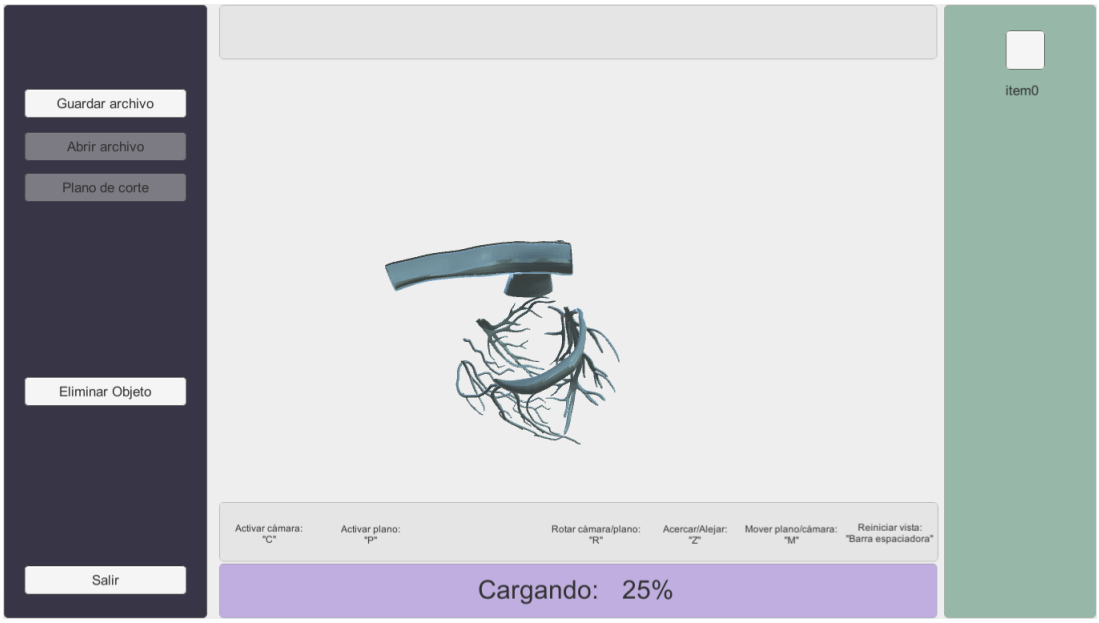

Figura 33: Interfaz en la que se muestran todos los cambios finales que se agregaron a la plataforma

Fuente: Elaboración propia

CAPÍTULO 13

Conclusiones

Los lenguajes de programación que se investigaron necesitan de herramientas de terceros para poder utilizar elementos gráficos, a diferencia de las plataformas de desarrollo que utilizan un lenguaje de programación como herramienta para modificar el comportamiento de sus elementos. Por lo que para un proyecto que tiene como objetivo el desarrollo de una aplicación, la mejor opción es seleccionar una plataforma de desarrollo como Unity.

La selección de una plataforma enfocada al desarrollo de aplicaciones para manejo de gráficos, simplificó en gran manera el proceso de diseño de la interfaz gráfica lo que permitió enfocarse en la implementación del código en vez de utilizar un mayor tiempo en los elementos gráficos como botones, acciones de los botones y el despliegue de información gráfica ya que Unity provee de elementos con comportamientos definidos.

Al evitar utilizar distintas librerías para el proyecto, facilitó la búsqueda de la información ya que Unity posee un manual en el que se explica el funcionamiento de cada uno de sus componentes. Esta característica facilitó la resolución del problema de los índices y el problema discutido en la Sección 11.1 donde la plataforma al momento de procesar objetos con muchos polígonos dejaba de responder.

La importación y exportación del archivo por medio de una librería que no utilizara el mismo tipo de arreglo que la plataforma de desarrollo aumentó la cantidad de operaciones que el programa necesitaba realizar antes de poder mostrar una imagen en pantalla. Conociendo el proceso completo desde la lectura del archivo hasta la exportación modificada se podría implementar una estructura de datos que se ajuste al que utiliza el renderizador para evitar el movimiento innecesario de información.

Durante las pruebas se identificó la necesidad de un algoritmo capaz de generar caras en el área de corte, este código debe poder identificar agujeros en el área expuesta por la sección del plano.

CAPÍTULO 14

Recomendaciones

- Se recomienda modificar el método utilizado para la importación y exportación de archivos STL por uno que almacene la información directamente a variables del tipo utilizado por Unity
- Para el proceso de corte se propone realizar una investigación acerca de algoritmos para triangulación de mallas que contengan agujeros, ya que por el tipo de objetos que se desean modificar estos contendrán estructuras internas que se deben mantener.
- Se recomienda dividir el total de información en varios hilos de ejecución para disminuir el tiempo de procesamiento dado que durante la carga de un archivo con gran cantidad de detalles, el tiempo esperado fue considerable.
- Se recomienda implementar la exportación del archivo STL modificado en otro hilo ya que durante la exportación de modelos con mayor detalle el proceso fue lento

CAPÍTULO 15

Bibliografía

[1] M. MacGill, *What should I expect during open heart surgery?*, Medical News Today, ago. de 2018. dirección: https://www.medicalnewstoday.com/articles/312888.php.

[2] F. Griesser, *What Resolution Can 3D Printers Print?*, oct. de 2016. dirección: https://all3dp.com/3d-printer-resolution/.

[3] S. Pieper, M. Halle y R. Kikinis, "3D Slicer", en *2004 2nd IEEE International Symposium on Biomedical Imaging: Macro to Nano (IEEE Cat No. 04EX821)*, IEEE. DOI: 10.1109/isbi.2004.1398617.

[4] A. Fedorov, R. Beichel, J. Kalpathy-Cramer, J. Finet, J.-C. Fillion-Robin, S. Pujol, C. Bauer, D. Jennings, F. Fennessy, M. Sonka, J. Buatti, S. Aylward, J. V. Miller, S. Pieper y R. Kikinis, "3D Slicer as an image computing platform for the Quantitative Imaging Network", *Magnetic Resonance Imaging*, vol. 30, n.º 9, págs. 1323-1341, ene. de 2012. DOI: 10.1016/j.mri.2012.05.001.

[5] B. Ripley, T. Kelil, M. K. Cheezum, A. Goncalves, M. F. D. Carli, F. J. Rybicki, M. Steigner, D. Mitsouras y R. Blankstein, "3D printing based on cardiac CT assists anatomic visualization prior to transcatheter aortic valve replacement", *Journal of Cardiovascular Computed Tomography*, vol. 10, n.º 1, págs. 28-36, ene. de 2016. DOI: 10.1016/j.jcct.2015.12.004.

[6] R. Dankowski, A. Baszko, M. Sutherland, L. Firek, P. Kałmucki, K. Wróblewska, A. Szyszka, A. Groothuis y T. Siminiak, "3D heart model printing for preparation of percutaneous structural interventions: description of the technology and case report", *Kardiologia Polska*, vol. 72, n.º 6, 2014. DOI: 10.5603/KP.2014.0119.

[7] R. Kikinis y S. Pieper, "3D Slicer as a tool for interactive brain tumor segmentation", en *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, IEEE, ago. de 2011. DOI: 10.1109/iembs.2011.6091765.

[8] H. Birthelmer, I. Soetebier y J. Sahm, "Efficient Representation of Triangle Meshes for Simultaneous Modification and Rendering", en *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2003, págs. 925-934. DOI: 10.1007/3-540-44860-8_96.

[9] U. Technologies, *Unity - Manual: Anatomy of a Mesh*, sep. de 2019. dirección: https://docs.unity3d.com/Manual/AnatomyofaMesh.html.

[10] J. de Vries, *LearnOpenGL - Face culling*, sep. de 2019. dirección: https://learnopengl.com/Advanced-OpenGL/Face-culling.

[11] D. Q. Nykamp, *The dot product*. dirección: https://mathinsight.org/dot_product.

[12] C. Iancu, D. Iancu y A. Stăncioiu, "FROM CAD MODEL TO 3D PRINT VIA"STL"FILE FORMAT.", *Fiability & Durability/Fiabilitate si Durabilitate*, n.º 1, 2010.

[13] A. Silberschatz, P. B. Galvin y G. Gagne, *Operating System Concepts*. Wiley, 2008, ISBN: 978-0470128725.

[14] J. Brodkin, *How Unity3D became a game-development beast*, jun. de 2013. dirección: https://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/.

[15] U. Technologies, *Powerful 2d - 3d software for modeling, animation, rendering, & simulation | App builder - Unity*, sep. de 2019. dirección: https://unity3d.com/unity#editor.

[16] ——, *Entity Component System | Package Manager UI website*, mayo de 2019. dirección: https://docs.unity3d.com/Packages/com.unity.entities@0.0/manual/index.html.

[17] ——, *Unity - Manual: Order of Execution for Event Functions*. dirección: https://docs.unity3d.com/Manual/ExecutionOrder.html.

[18] D. Poole, *Algebra Lineal (English and Spanish Edition)*. Cengage Learning Editores, 2011, ISBN: 978-607-481-608-2.

[19] J. D. Foley, S. K. Feiner, J. F. Hughes y A. V. Dam, *Computer Graphics: Principles and Practice*. Addison-Wesley, 1990, ISBN: 0-201-12110-7.

[20] B. Pregun, "Three-Dimensional Mesh Cutting in Virtual Scene", Tesis de mtría., University of Zagreb, 2017.

[21] J. Mitani, *A Simple-to-Implementation Method for Cutting a Mesh Model by a Hand-Drawn Stroke*, eng, 2005. DOI: 10.2312/sbm/sbm05/035-041.

[22] U. Technologies, *Unity - Scripting API: IndexFormat*, sep. de 2019. dirección: https://docs.unity3d.com/ScriptReference/Rendering.IndexFormat.html.

[23] C. Garcia, *P15 - Vented and Drained Succulent Planter*, jul. de 2018. dirección: https://www.thingiverse.com/thing:3024364.

[24] T. Hill, *Planttlement - Castle Planter with Moat, for Succulents*, jul. de 2018. dirección: https://www.thingiverse.com/thing:3024153.

[25] B. Locicero, *Anatomical Heart*, jul. de 2015. dirección: https://www.thingiverse.com/thing:932606.

[26] D. O'Connell, *Sphericon*, 2017. dirección: https://www.myminifactory.com/object/3d-print-sphericon-46248.

CAPÍTULO 16

Anexos

Repositorio donde se encuentra el código del proyecto:

`https://github.com/aju13171/Plataforma-modificacion-STL.git`